Définition Formelle — Glushkovizer

Résumé

Ce document constitue la définition formelle des différents types de données utilisés tout au long de cette librairie. Dans un premier temps, nous nous concentrerons sur les expressions régulières et les fonctions définies sur celles-ci. Puis, dans un second temps, nous nous intéresserons aux automates et à leurs diverses fonctions définies sur eux. Enfin, pour finir, nous parlerons d'automates particuliers, ceux de Glushkov, nous aborderons leurs constructions ainsi que leurs propriétés.

Table des matières

I Prélude **3**
1 Les mots . . . 3
2 Les langages . . . 4
3 Conclusion . . . 5

II Les expressions régulières **6**
1 Définition . . . 6
2 Fonction sur les *ER* . . . 7
3 Conclusion . . . 11

III Les automates **13**
1 Définition . . . 13
2 Fonction sur les automates . . . 18
3 Conclusion . . . 18

IV Les automates de Glushkov **20**
1 Définition . . . 20
2 Propriétés : . . . 21
3 Conclusion . . . 22

V Conclusion **23**

I Prélude

Pour la compréhension de l'ensemble de ce document, nous avons besoin de plusieurs notions de théorie des langages. C'est donc pourquoi dans cette partie, nous allons étudier les différentes notions nécessaires. Dans un premier temps, nous allons définir ce qu'est un mot et quelles sont les opérations sur les mots. Enfin, dans un second temps, nous définirons ce qu'est un langage et quelles opérations sont munies par les langages.

1 Les mots

Définition 1.1. Un *alphabet* Σ est un ensemble fini non-vide de symboles. Un *mot* est une suite finie de symboles sur un alphabet Σ. Le mot composé de zéro symbole est appelé mot vide et est noté ε.

Example 1.1.1.

$$\Sigma = \{a, b, c, d\}$$
$$w = abbcdda$$

Définition 1.2. On parlera de la *longueur d'un mot* w noté $|w|$ pour désigner le nombre de symboles qui le composent.

Example 1.2.1.

$$w = abbcdda$$
$$|w| = 7$$

Définition 1.3. Une des opérations sur les mots est la concaténation de mots. On notera la *concaténation* de deux mots $u = a_1 \cdots a_n$ et $v = b_1 \cdots b_n$ par $u \cdot v$. Qui est ainsi égal à $u \cdot v = a_1 \cdots a_n b_1 \cdots b_n$. On définit l'ensemble des mots sur Σ par Σ^*. On notera que :

- La concaténation est associative $(w \cdot u) \cdot v = w \cdot (u \cdot v)$.
- La concaténation admet un élément neutre $u \cdot \varepsilon = \varepsilon \cdot u = u$.

Ce qui implique que l'ensemble Σ^* muni de la concaténation $(\Sigma, \cdot)$ forme un monoïde.

Example 1.3.1.

$$u = abab$$
$$v = cdcd$$
$$u \cdot v = ababcdcd$$

2 Les langages

Définition 1.4. Un *langage* L est un ensemble de mots sur un alphabet fini Σ. On appellera *langage vide* le langage ne comportant aucun mot, ainsi défini comme ceci : $L = \varnothing$.

Example 1.4.1.

$$\Sigma = \{a, b, c, d\}$$
$$L_1 = \{a, aa, bc, da, \varepsilon\}$$
$$L_2 = \varnothing$$

Définition 1.5. L'une des opérations sur les langages est l'*union*. On parlera de l'union de langage notée $L_1 \cup L_2$ et définie comme ceci :

$$L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$$

On notera que l'union est associative, commutative et admet un élément neutre ($\varnothing$).

Example 1.5.1.

$$\Sigma = \{a, b, c, d\}$$
$$L_1 = \{\varepsilon, a, aa, bc, da\}$$
$$L_2 = \{d, aa, cd\}$$
$$L_1 \cup L_2 = \{\varepsilon, a, d, aa, cd, bc, da\}$$

Définition 1.6. Une autre opération sur les langages est la *concaténation*. Elle est définie en utilisant la concaténation des mots qui composent les langages. Cette opération est ainsi définie comme ceci :

$$L_1 \cdot L_2 = \{u \cdot v \mid u \in L_1, v \in L_2\}$$

On remarquera qu'elle est associative, pas commutative et admet un élément neutre ($\{\varepsilon\}$). Et que $\varnothing$ est aussi un élément absorbant pour cette opération.

Example 1.6.1.

$$\Sigma = \{a, b, c, d\}$$
$$L_1 = \{\varepsilon, a, aa\}$$
$$L_2 = \{d, cc\}$$
$$L_1 \cdot L_2 = \{d, ad, aad, cc, acc, aacc\}$$

Définition 1.7. Par extension, on définit la *copie n-ième* d'un langage L notée L^n et définit récursivement comme ceci :

$$L^0 = \{\varepsilon\}$$
$$L^n = L^{n-1} \cdot L$$

On remarquera que $\varnothing^0 = \{\varepsilon\}$.

Example 1.7.1.

$$\Sigma = \{a, b\}$$
$$L = \{\varepsilon, a\}$$
$$L^3 = \{\varepsilon, a, aa, aaa\}$$

Définition 1.8. Grâce à cette opération, on peut définir l'*étoile* d'un langage notée L^*. Qui peut être définie comme ceci :

$$L^* = \bigcup_{i \geq 0} L^i$$

Example 1.8.1.

$$L = \{a\}$$
$$L^* = \{\varepsilon\} \cup \{a\} \cup \{aa\} \cup \cdots$$

3 Conclusion

Nous avons défini les concepts de *mot*, de *langage* et l'ensemble des opérations applicables à ces objets. Bien que nous n'ayons couvert qu'une partie des opérations possibles, nous vous encourageons à consulter un cours de théorie des langages pour obtenir des informations plus détaillées. Nous vous conseillons ces ressources : [Har78], [Aut94] et [HMU07].

II Les expressions régulières

Dans cette section, nous parlerons d'expressions régulières (ER). Nous allons nous concentrer sur un type bien particulier d'expressions régulières qui ne seront pas les expressions régulières que nous pouvons voir plus quotidiennement dans le domaine de l'informatique, les expressions régulières *UNIX*. Mais plutôt une version plus simple de celles-ci.

1 Définition

Nous allons noter une expression régulière $E \in Exp(\Sigma)$, c'est-à-dire une expression régulière où les symboles sont inclus dans l'ensemble Σ et où $Exp(\Sigma)$ représente l'ensemble des expressions sur Σ. Cette expression reconnait un langage qu'on pourra appeler $L(E)$. Nous pouvons définir une expression régulière récursivement de cette manière :

$$E = \varepsilon \tag{1}$$
$$E = a \tag{2}$$
$$E = F + G \tag{3}$$
$$E = F \cdot G \tag{4}$$
$$E = F^* \tag{5}$$
$$E = (F) \tag{6}$$

avec E, F et G des expressions régulières sur Σ et a un symbole de Σ

On notera que $*$ est prioritaire sur $\cdot$ qui est lui-même prioritaire sur $+$ et qu'ils sont tous deux associatifs à gauche. On comprend donc pourquoi l'équation (6) existe, elle est là pour des raisons de priorité. Il est alors évident de calculer les diverses fonctions sur celle-ci, c'est pour cela qu'on ne précisera pas son calcul. On peut définir chaque équation comme ceci :

- $E = \varepsilon$ **:** Représente le mot vide, de ce fait un mot de longueur zéro. Il peut être parfois représenté par « $ ».
- $E = a$ **:** Représente un symbole présent dans l'ensemble Σ
- $E = F + G$ **:** Représente l'union des deux expressions régulières F et G. Par abus de langage, on peut aussi dire F « ou » G pour représenter cette union.
- $E = F \cdot G$ **:** Représente la concaténation des deux expressions régulières F et G.

- $E = F^*$ **:** Représente l'union infinie de copie de F, cette répétition incluant la puissance zéro et donc le mot vide.

Pour calculer le langage que dénote l'expression régulière, on peut le calculer récursivement de cette manière :

$$L(\varepsilon) = \{\varepsilon\}$$
$$L(a) = \{a\}$$
$$L(F + G) = L(F) \cup L(G)$$
$$L(F \cdot G) = L(F) \cdot L(G)$$
$$L(F^*) = (L(F))^*$$

avec F et G des expressions régulières sur Σ et a un symbole de Σ

Example 2.0.1. On comprendra ainsi que l'expression $E = a + c \cdot d$ avec $E \in Exp(\Sigma)$ et $\Sigma = \{a, b, c, d\}$, dénote le langage $L(E) = \{a, cd\}$. Car on peut représenter E comme ceci :

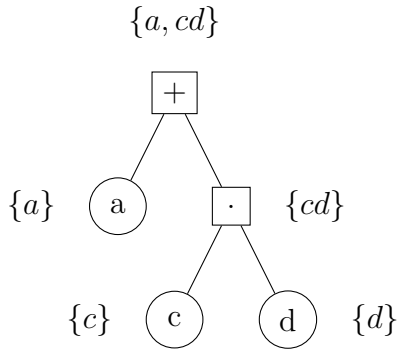


FIGURE 1 – Représentation de l'expression régulière à l'aide d'un arbre syntaxique

Comme on peut voir sur la Figure 1, grâce à cette représentation, on peut calculer simplement le langage reconnu par l'expression régulière (ici représenté par les ensembles à côté de chaque arbre).

2 Fonction sur les *ER*

Plusieurs informations sur les expressions régulières nous seront utiles, comme l'ensemble des premiers/derniers symboles des mots du langage décrit par l'expression. Il serait aussi intéressant de savoir si son langage contient le mot vide. Et d'avoir les successeurs des symboles, c'est-à-dire les symboles suivant un symbole donné.

On pourrait calculer individuellement chaque information, mais nous pouvons calculer tout d'un coup avec une fonction qu'on pourrait appeler $flnf$. Elle permet

de calculer un tuple contenant toutes ces informations pour une expression régulière donné.

On aurait donc pour une expression régulière E sur l'alphabet Σ, ceci :

$$flnf(E) = (F, L, \Theta, \delta)$$

- $F \subseteq \Sigma$: Ensemble des premiers symboles de l'expression régulière
- $L \subseteq \Sigma$: Ensemble des derniers symboles de l'expression régulière
- $\Theta = \begin{cases} \{\varepsilon\}, & \text{si } \varepsilon \in L(E) \\ \varnothing & \text{sinon} \end{cases}$
- $\delta : \Sigma \to 2^{\Sigma}$ fonction renvoyant les successeurs du symbole donné

La fonction $flnf$ a donc comme signature :

$$flnf : Exp(\Sigma) \to (2^{\Sigma} \times 2^{\Sigma} \times \{\varnothing, \{\varepsilon\}\} \times \Sigma \to 2^{\Sigma})$$

Et peut-être calculée de cette manière, pour E et G des expressions régulière sur l'alphabet Σ et a un symbole de Σ :

$$flnf(\varepsilon) = (\varnothing, \varnothing, \varepsilon, \delta) \mid \delta(a) = \varnothing, a \in \Sigma$$

$$flnf(a) = (\{a\}, \{a\}, \varnothing, \delta) \mid \delta(a) = \varnothing, a \in \Sigma$$

$$\begin{gathered} flnf(E + G) = (F \cup F', L \cup L', \Theta \cup \Theta', \delta'') \text{ avec} \\ \delta''(a) = \delta(a) \cup \delta'(a) \mid \forall a \in \Sigma \\ (F, L, \Theta, \delta) = flnf(E) \wedge (F', L', \Theta', \delta') = flnf(G) \end{gathered}$$

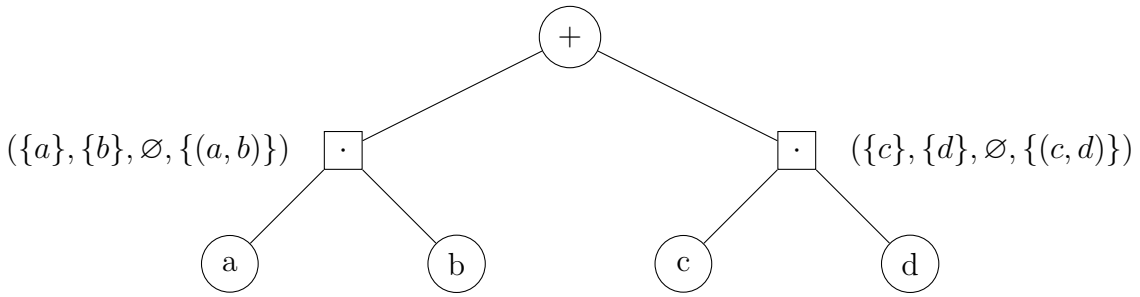
$$\begin{gathered} flnf(E \cdot G) = (F'', L'', \Theta \cap \Theta', \delta'') \text{ avec} \\ F'' = F \cup F' \cdot \Theta \\ L'' = L' \cup L \cdot \Theta' \\ \delta''(a) = \begin{cases} \delta(a) \cup \delta'(a) \cup F', & \text{si } a \in L \\ \delta(a) \cup \delta'(a) & \text{sinon} \end{cases} \mid \forall a \in \Sigma \\ (F, L, \Theta, \delta) = flnf(E) \wedge (F', L', \Theta', \delta') = flnf(G) \end{gathered}$$

$$flnf(E^*) = (F, L, \{\varepsilon\}, \delta') \text{ avec}$$
$$\delta'(a) = \begin{cases} \delta(a) \cup F, & \text{si } a \in L \\ \delta(a) & \text{sinon} \end{cases} \mid \forall a \in \Sigma$$
$$(F, L, \Theta, \delta) = flnf(E)$$

Remarque 2.1. Il existe un isomorphisme entre les fonctions et les couple antécédents, images. Ce qui fait que la fonction des successeurs pourra être représenté à l'aide d'un couple.

Example 2.1.1. Prenons par exemple l'expression régulière suivante $E = a \cdot b + c \cdot d$, avec $E \in Exp(\Sigma)$ et $\Sigma = \{a, b, c, d\}$. Toujours à l'aide d'un arbre syntaxique, on peut calculer ce que $flnf(E)$ donnerait.

$$(\{a, c\}, \{b, d\}, \varnothing, \{(a, b), (c, d)\})$$



$$(\{a\}, \{a\}, \varnothing, \varnothing) \quad (\{b\}, \{b\}, \varnothing, \varnothing) \quad (\{c\}, \{c\}, \varnothing, \varnothing) \quad (\{d\}, \{d\}, \varnothing, \varnothing)$$

FIGURE 2 – Représentation de l'expression régulière à l'aide d'un arbre syntaxique.

Il advient que $flnf(E) = \{\{a, c\}, \{b, d\}, \varnothing, \delta\}$ avec δ qui est défini comme ceci :

$$\delta(a) = \{b\}$$
$$\delta(b) = \varnothing$$
$$\delta(c) = \{d\}$$
$$\delta(d) = \varnothing$$

Example 2.1.2. Un autre exemple pourrait être $E' = (a + b) \cdot c^*$, avec cet exemple, on voit l'utilité de la parenthèse, car sans elle la concaténation aurait été sur $b \cdot c^*$. Et comme dit précédemment (1), son calcul revient à calculer l'expression contenue entre les parenthèses.

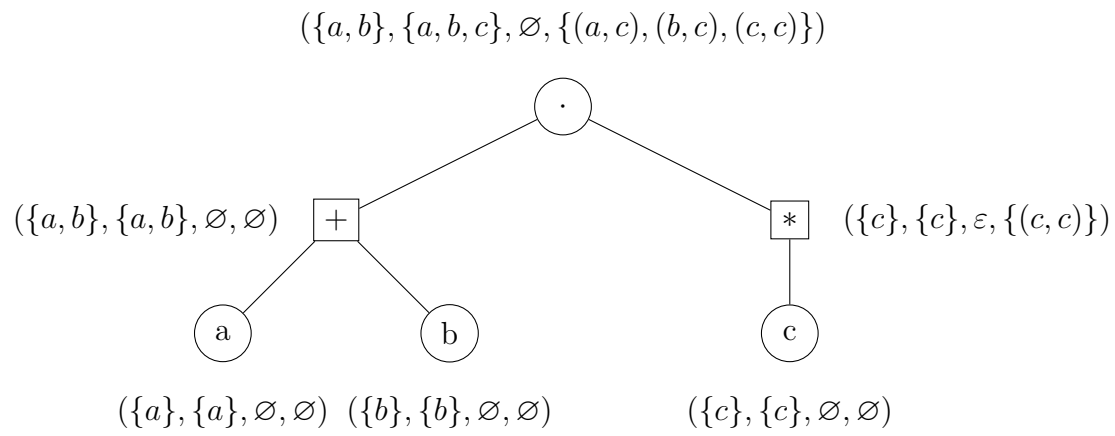


FIGURE 3 – Représentation de l'expression régulière à l'aide d'un arbre syntaxique.

Ce qui fait que $flnf(E') = (\{a,b\},\{a,b,c\},\varnothing,\delta')$ avec δ' qui est défini comme décrit après :

$$\begin{aligned}\delta'(a) &= \{c\}\\ \delta'(b) &= \{c\}\\ \delta'(c) &= \{c\}\\ \delta'(d) &= \varnothing\end{aligned}$$

Une autre fonction qui s'applique aux expressions régulières est *linearization* ; (elle peut paraitre inutile, mais) elle nous servira dans la Section IV. Sa signature est :

$$linearization : Exp(\Sigma) \to Exp(\Sigma \times \mathbb{N})$$

Elle peut être définie de cette manière, pour $a \in \Sigma$ et $(E,F) \in (Exp(\Sigma))^2$:

$$linearization(E) = \pi_2(linearization\_aux(E,1)) \quad \text{avec}$$

Avec π_n la fonction de projection sur les tuples et $linearization\_aux$ définie récursivement comme ceci :

$$
\begin{aligned}
&linearization\_aux(\varepsilon, n) = (\varepsilon, n) \\
&linearization\_aux(a, n) = ((a, n), n + 1) \\
&linearization\_aux(E + F, n) = (E' + F', n'') \quad \text{avec} \\
&(E', n') \leftarrow linearization\_aux(E, n) \\
&(F', n'') \leftarrow linearization\_aux(F, n') \\
&linearization\_aux(E \cdot F, n) = (E' \cdot F', n'') \quad \text{avec} \\
&(E', n') \leftarrow linearization\_aux(E, n) \\
&(F', n'') \leftarrow linearization\_aux(F, n') \\
&linearization\_aux(E^*, n) = (E'^*, n') \quad \text{avec} \\
&(E', n') \leftarrow linearization\_aux(E, n)
\end{aligned}
$$

Avec cette définition, on peut voir que tous les symboles sont associés à un unique entier. Ce qui fait que l'expression régulière résultante ne contient que des symboles uniques. Et que, de ce fait, si deux couples partagent le même entier, cela implique qu'ils ont la même valeur de symbole.

Example 2.1.3. Si on prend l'expression régulière $E = \varepsilon + b^* \cdot b$, avec $E \in Exp(\Sigma)$ et $\Sigma = \{a, b, c, d\}$.

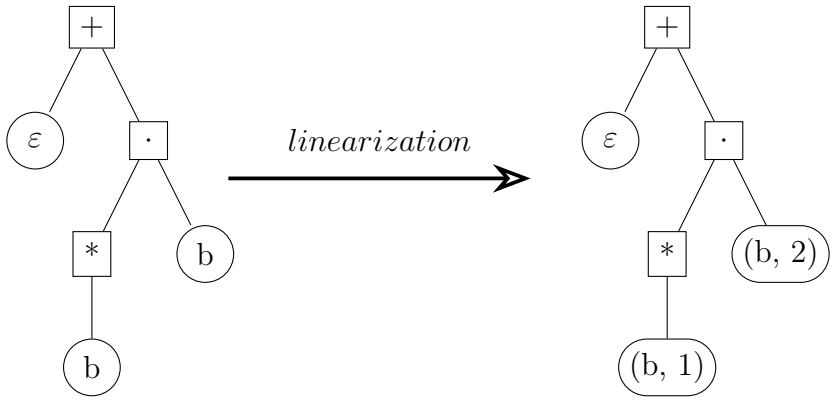


FIGURE 4 – Représentation à l'aide d'un arbre syntaxique de l'expression régulière une fois après avoir fait appel à *linearization* sur elle.

3 Conclusion

On saisit aisément que ces expressions ont beau être simples (peu d'opération comparé aux expressions régulières d'*UNIX*). On peut voir qu'elles permettent de décrire des langages très complexes et en quantité infinie. En revanche, il est difficile de

savoir si un mot est reconnu par une expression régulière simplement. Par exemple est-ce que le mot *eipipipipipip* est reconnu par cette expression $((((o \cdot \varepsilon) + (\varepsilon \cdot e)) + ((g \cdot \varepsilon) \cdot \varepsilon^*)) \cdot ((\varepsilon \cdot i) \cdot (p + \varepsilon))^*)$? La réponse est oui. C'est pour cela qu'il serait peut-être intéressant d'utiliser un autre objet pour reconnaitre des mots, comme les automates que nous allons voir maintenant.

III Les automates

Dans cette partie, nous parlerons des automates et plus particulièrement, nous allons parler des automates sans ε-transition (des automates utilisent des ε-transitions, comme ceux de *Thompson* [Tho68], qui sont utilisés par nos ordinateurs). Pour autant, les automates que nous verrons ne sont pas limités par le manque de ces transitions.

1 Définition

Comme dit précédemment, un automate est un objet mathématique reconnaissant un langage. On notera $M \in AFN(\Sigma, \eta)$ l'automate qui a pour transition des valeurs dans Σ, des valeurs « d'état » dans η. Et $AFN(\Sigma, \eta)$ l'ensemble des automates finis non déterministes de valeur de transition dans Σ et de valeur d'état dans η. On écrira $L(M)$ pour désigner le langage qu'il reconnait. Un automate est un tuple qu'on peut écrire de cette forme $M = (Q, I, F, \delta)$ avec :

$$
\begin{aligned}
Q &\subseteq \eta && \text{L'ensemble des états qui constituent l'automate} \\
I &\subseteq Q && \text{L'ensemble des états initiaux} \\
F &\subseteq Q && \text{L'ensemble des états finaux} \\
\delta &: Q \times \Sigma \to 2^Q && \text{La fonction de transition}
\end{aligned}
$$

Un automate peut se représenter à l'aide d'un graphe orienté, valué, particulier. Par exemple si on veut représenter $M = (\{q_1, q_2, q_3, q_4, q_5\}, \{q_1\}, \{q_2, q_3\}, \delta)$ avec $M \in AFN(\Sigma, \eta)$, $\Sigma = \{0, 1\}$, $\eta = \{q_1, q_2, q_3, q_4, q_5\}$ et δ défini comme ceci :

$$
\begin{aligned}
\delta(q_1, 0) &= \{q_2, q_4\} & \qquad \delta(q_3, 1) &= \{q_4\} \\
\delta(q_1, 1) &= \varnothing & \qquad \delta(q_4, 0) &= \{q_5\} \\
\delta(q_2, 0) &= \varnothing & \qquad \delta(q_4, 1) &= \{q_3\} \\
\delta(q_2, 1) &= \varnothing & \qquad \delta(q_5, 0) &= \{q_4\} \\
\delta(q_3, 0) &= \{q_3\} & \qquad \delta(q_5, 1) &= \{q_5\}
\end{aligned}
$$

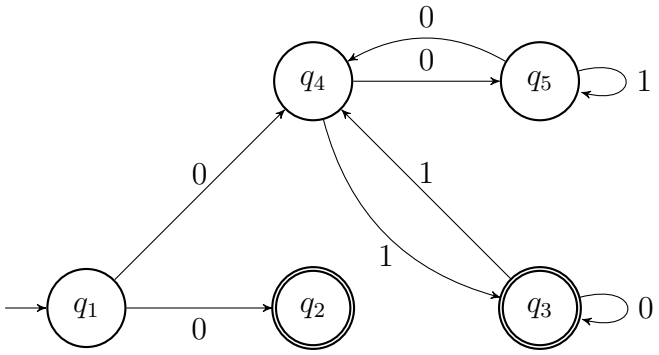


FIGURE 5 – Exemple de représentation graphique d'un automate.

Dans la Figure 5, on peut voir que les états initiaux (dans cet automate n'y a qu'un seul initial ; q_1) ont une petite flèche qui pointe sur eux et que les états finaux ont un double contour. Et que les transitions sont symbolisées par des flèches entre les états et que ces flèches sont labellisées.

On peut aussi étendre la fonction de transition δ de manière qu'elle ait comme signature :

$$\delta : Q \times \Sigma^* \to 2^Q$$

En la définissant récursivement de telle sorte :

$$\begin{aligned} \delta(q, \varepsilon) &= \{q\} \\ \delta(q, a \cdot w) &= \bigcup_{q' \in \delta(q,a)} \delta(q', w) \quad \text{avec } a \in \Sigma \end{aligned}$$

Example 3.0.1. Voici donc quelques exemples si on prend l'automate utilisé pour la représentation graphique (Figure 5) :

$$\begin{aligned} \delta(q_1, 00) &= \{q_5\} \\ \delta(q_1, 11) &= \varnothing \\ \delta(q_1, \varepsilon) &= \{q_1\} \\ \delta(q_1, 00 \cdot 1^n) &= \{q_5\} \text{ avec } n \in \mathbb{N} \end{aligned}$$

Définition 3.1. Un automate est dit *standard* quand il ne possède qu'un seul état initial non ré-entrant, aussi défini comme ceci :

$$M = (Q, \{i\}, F, \delta) \quad \text{avec}$$
$$\forall p \in Q, \forall a \in \Sigma \mid i \notin \delta(p, a)$$
$$M \in AFN(\Sigma, \eta)$$

Définition 3.2. Un automate est *homogène* lorsque, pour tous les états, les transitions allant vers cet état ont la même valeur. En d'autres termes, quand il respecte cette propriété :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$\forall (p, q, r) \in Q^3, \exists (a, b) \in \Sigma^2 \mid q \in \delta(p, a) \wedge q \in \delta(r, b) \Longrightarrow a = b$$
$$M \in AFN(\Sigma, \eta)$$

Définition 3.3. Un automate est qualifié d'*accessible* lorsqu'en partant des initiaux, on peut arriver sur tous les états qui le composent. C'est-à-dire qu'il valide cette condition :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$\forall p \in Q, \exists w \in \Sigma^* \mid p \in \bigcup_{i \in I} \delta(i, w)$$
$$M \in AFN(\Sigma, \eta)$$

Définition 3.4. Un automate est considéré comme *coaccessible* dès que, de tous les états, on peut arriver à un état final. Ceci veut dire qu'il atteste de cette particularité :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$\forall p \in Q, \exists w \in \Sigma^* \mid F \cap \delta(p, w) \neq \varnothing$$
$$M \in AFN(\Sigma, \eta)$$

Définition 3.5. Un automate est dit *déterministe* quand tous ses états vont au maximum à un état par symbole et que l'automate ne possède qu'un seul état initial. Autrement dit qu'il valide cette propriété :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$|I| = 1 \wedge \forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| \leq 1$$
$$M \in AFN(\Sigma, \eta)$$

On parlera de *déterministe complet* lorsque tous ses états vont sur un état par symbole. C'est-à-dire qu'il respecte cette condition :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$|I| = 1 \wedge \forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| = 1$$
$$M \in AFN(\Sigma, \eta)$$

Example 3.5.1. Donc, l'automate représenté sur la Figure 5 est standard, non homogène, accessible et coaccessible. Car il possède bien un unique état initial (q_1), mais q_3, q_4 et q_5 ne respecte pas la propriété pour être homogène, parce qu'ils ont des transitions allant vers eux avec des valeurs différentes. De plus, tous ses états sont accessibles depuis l'état initial. Et son inverse est, lui-même aussi, accessible et il n'est pas déterministe.

Définition 3.6. Nous parlerons de sous-automate pour parler d'une « région » d'un automate. N est un sous automate de M qu'on notera $N \subseteq M$, s'il vérifie cette propriété :

$$N = (Q', I', F', \delta') \quad \text{avec}$$
$$Q' \subseteq Q \wedge I' \subseteq Q' \wedge F' \subseteq Q'$$
$$\delta' \text{ est une restriction de } \delta, \delta' : Q' \to 2^{Q'}$$
$$M = (Q, I, F, \delta) \wedge (M, N) \in (AFN(\Sigma, \eta))^2$$

Les automates pouvant être représentés à l'aide de graphes, on peut étendre les propriétés sur les graphes aux automates. Par exemple, on pourra parler des composantes fortement connexes d'un automate. Autrement dit, en partant de n'importe quel état, on peut arriver à tous les autres états. Ainsi, ça veut dire qu'un automate fortement connexe vérifierait ceci :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$\forall (p, q) \in Q^2, \exists w \in \Sigma^* \mid q \in \delta(p, w)$$
$$M \in AFN(\Sigma, \eta)$$

Définition 3.7. Une autre notion qui est présente sur les graphes que nous allons adapter sur les automates est la notion de *hamac*. Nous dirons qu'un automate est un *hamac* lorsqu'il est standard, accessible et coaccessible (nous gardons le nom *hamac* pour une raison de compréhension). Ceci veut dire qu'il peut être décrit comme ceci :

$$M = (Q, I, F, \delta) \quad \text{avec}$$
$$standard(M) \wedge accessible(M) \wedge coaccessible(M)$$
$$M \in AFN(\Sigma, \eta)$$

Définition 3.8. Une autre idée empruntée au graphe est la notion d'*orbite*. Nous dirons qu'un sous-automate est une *orbite*, si pour tout couple d'état i et t, il existe un mot non vide permettant d'aller de i à t. Aussi défini comme ceci :

$$\mathcal{O} = (Q, I, F, \delta) \quad \text{avec}$$
$$\forall (p, q) \in Q^2, \exists w \in \Sigma^* \setminus \{\varepsilon\} \mid q \in \Sigma(p, w)$$
$$\mathcal{O} \subseteq M \wedge M \in AFN(\Sigma, \eta)$$

Dans la même idée, nous parlerons d'*orbite maximale* lorsque l'orbite n'est incluse dans aucune orbite différente. En d'autres termes, que l'orbite est fortement connexe sans prendre les chemins triviaux (mot vide).

Définition 3.9. Nous noterons $In(\mathcal{O})$ et $Out(\mathcal{O})$ respectivement l'ensemble des portes d'entrée et l'ensemble des portes de sortie de l'orbite $\mathcal{O}$. Qui sont définies de cette façon :

$$In(\mathcal{O}) = \{p \in Q' \mid \exists a \in \Sigma, \exists q \in Q \setminus Q', p \in \delta(q, a)\} \cup I'$$
$$Out(\mathcal{O}) = \{p \in Q' \mid \exists a \in \Sigma, \exists q \in Q \setminus Q', q \in \delta(p, a)\} \cup F'$$
$$\text{avec}$$
$$\mathcal{O} = (Q', I', F', \delta') \wedge M = (Q, I, F, \delta)$$
$$\mathcal{O} \subseteq M \wedge M \in AFN(\Sigma, \eta)$$

Définition 3.10. Avec ceci, on peut définir ce qu'est une *orbite stable*. Une orbite est dite *stable* quand pour toutes les sorties, il existe une transition vers toutes les entrées. C'est-à-dire que l'orbite vérifie ceci :

$$\forall q \in Out(\mathcal{O}), \exists a \in \Sigma \mid \delta(q, a) \cap In(\mathcal{O}) \neq \varnothing$$
$$\text{avec}$$
$$\mathcal{O} = (Q, I, F, \delta) \subseteq M \wedge M \in AFN(\Sigma, \eta)$$

On la qualifiera même de *fortement stable* lorsqu'en supprimant toutes les transitions de portes des sorties vers les portes d'entrée, les orbites maximales de l'orbite sont stables et *fortement stables*.

Définition 3.11. De même, on dira qu'une orbite est *transversale* si toutes les entrées viennent des mêmes états et que toutes les sorties vont aux mêmes états. Autrement dit, que l'orbite valide cette propriété :

$$\forall (p, q) \in (Out(\mathcal{O}))^2, (\bigcup_{a \in \Sigma} \delta(p, a)) \cap Q \setminus Q' = (\bigcup_{a \in \Sigma} \delta(q, a)) \cap Q \setminus Q'$$
$$\forall (p, q) \in (In(\mathcal{O}))^2, \{r \in Q \setminus Q' \mid \exists a \in \Sigma, p \in \delta(r, a)\} = \{r \in Q \setminus Q' \mid \exists a \in \Sigma, q \in \delta(r, a)\}$$
$$\text{avec}$$
$$\mathcal{O} = (Q', I', F', \delta) \subseteq M = (Q, I, F, \delta) \wedge M \in AFN(\Sigma, \eta)$$

Elle sera même *fortement transversale* lorsqu'en supprimant toutes les transitions de portes des sorties vers les portes d'entrée, les orbites maximales de l'orbite sont transversales et *fortement transversales*.

2 Fonction sur les automates

Une des fonctions sur les automates est *accept* qui vérifie si le mot est reconnu par l'automate. C'est-à-dire que si on prend le chemin décrit par le mot donné en argument, on arrive sur un ou plusieurs états finaux. Elle a alors pour signature :

$$accept : AFN(\Sigma, \eta) \times \Sigma^* \to \mathbb{B}$$

Elle peut être définie simplement comme ceci :

$$accept(M, w) = (\bigcup_{p \in I} \delta(p, w)) \cap F \neq \varnothing$$

Une autre fonction sur les automates est *homogenized* qui renvoie l'automate homogène qui reconnait le même langage que l'automate donné. Elle a ainsi comme signature :

$$homogenized : AFN(\Sigma, \eta) \to AFN(\Sigma, (\Sigma \cup \{\varepsilon\} \times \eta))$$

Elle peut être définie de cette façon :

$$\begin{gathered}
homogenized(M) = N \quad \text{avec} \\
\forall (p, q) \in Q^2, \exists a \in \Sigma \mid p \in \delta(q, a) \Rightarrow (a, p) \in Q' \\
\forall (p, q) \in Q^2, \forall a \in \Sigma \mid p \notin \delta(q, a) \Rightarrow (\varepsilon, p) \in Q' \\
\forall p \in I, \forall a \in \Sigma \cup \{\varepsilon\} \mid (a, p) \in Q' \Rightarrow (a, p) \in I' \\
\forall p \in F, \forall a \in \Sigma \cup \{\varepsilon\} \mid (a, p) \in Q' \Rightarrow (a, p) \in F' \\
\forall (p, q) \in Q, \exists a \in \Sigma, \forall b \in \Sigma \cup \{\varepsilon\} \mid (b, q) \in Q', p \in \delta(q, a) \Rightarrow (p, a) \in \delta'((b, q), a) \\
M = (Q, I, F, \delta) \in AFN(\Sigma, \eta) \\
N = (Q', I', F', \delta') \in AFN(\Sigma, (\Sigma \cup \{\varepsilon\} \times \eta))
\end{gathered}$$

On remarque bien que par construction l'automate résultant est homogène, parce que les états sont devenus un couple entre leur valeur et leur transition entrante. Ce qui fait que toutes les transitions vers l'état (a, p) ont tous pour valeur a.

3 Conclusion

Comme nous venons de voir, les automates sont des outils pour reconnaitre des mots d'un langage. L'une de leurs grandes forces est leur simplicité. Toutes les opérations sur les automates peuvent donc être automatisées. Ce qui fait que cet objet est

très intéressant dans le monde de l'informatique. En revanche, l'un de ses points faibles est que pour nous humain, il est difficile de représenter un automate autrement que par une représentation graphique. Contrairement aux expressions régulières. Il serait alors intéressant de pouvoir convertir une expression régulière en automate. On pourrait se poser la question « est-ce-que c'est toujours possible de convertir une expression régulière en automate », la réponse est oui, car selon le théorème de Kleene [Kle51], toute expression régulière peut être représentée par un automate fini. Nous allons voir un algorithme pour faire cette conversion dans la prochaine section.

IV Les automates de Glushkov

Le terme « automate de Glushkov » est un abus de langage, faisant référence aux automates que l'algorithme de transformation d'expression régulière en automate, appelé algorithme de Glushkov produit. Son nom vient de l'informaticien soviétique Victor Glushkov qui est son créateur [Glu61].

1 Définition

Nous appliquerons cet algorithme à l'aide de la fonction *glushkov* qui a donc pour signature :

$$glushkov : Exp(\Sigma) \to AFN(\Sigma, \mathbb{N})$$

Et a ainsi, on peut définir cette fonction de cette façon :

$$\begin{aligned} glushkov(E) &= (Q, \{0\}, F, \delta) \quad \text{avec} \\ Q &\leftarrow \{n \mid n \in \mathbb{N} \wedge 0 \leq n < m\} \\ F &\leftarrow \{n \mid (a, n) \in Last\} \cup (\{0\} \cdot Null) \\ \forall q \in \delta(p, a) \mid & \begin{cases} (a, q) \in First, & \text{si } p = 0 \\ (a, q) \in Follow((b, p)) & \text{sinon} \end{cases} \\ (E', m) &\leftarrow linearization(E) \\ (First, Last, Null, Follow) &\leftarrow flnl(E') \end{aligned}$$

Example 4.0.1. Vu qu'un dessin vaut toujours mieux que mille mots, voici un exemple de l'automate résultant de la transformation de cette expression $E = (a + b) \cdot a^* \cdot b^* \cdot (a + b)^*$.

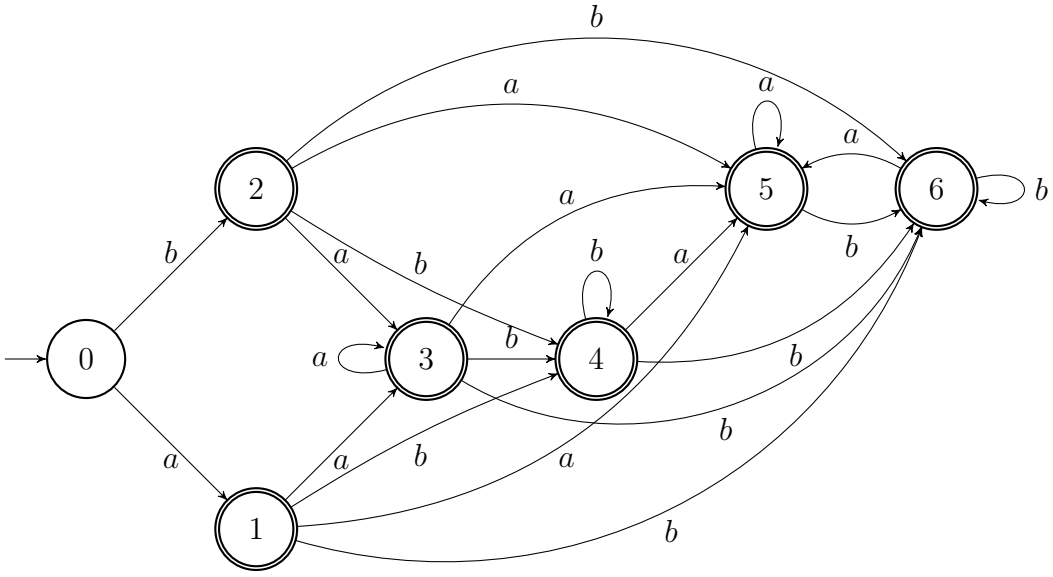


FIGURE 6 – Exemple de représentation graphique de l'automate résultant de $glushkov(E)$.

On peut remarquer qu'il y a des propriétés intéressantes sur cet automate. C'est ce que l'on va étudier maintenant.

2 Propriétés :

Nous verrons ici plusieurs propriétés sur les automates de Glushkov, mais nous n'en ferons pas la preuve, nous en donnerons une justification, mais pas une réelle preuve (preuve disponible dans ce document [CZ00]).

1. Les automates de Glushkov sont *standards*, car par construction, il ne peut avoir qu'un seul état initial (0) non ré-entrant.

2. L'automate a $n+1$ avec n le nombre de symboles de l'expression régulière. Le $+1$ vient du fait que nous ajoutons un état 0 qui a des transitions vers les $First$.

3. Les automates de Glushkov sont accessibles et coaccessibles. C'est dû au fait que chaque symbole dans l'expression régulière est accessible et coaccessible et que cette propriété ne se perd pas lors de la transformation.

4. L'automate de Glushkov est homogène. Cela résulte de sa construction, car pour qu'un état aille sur un autre état, il faut qu'il ait dans ses $Follow$ (a, n) avec a le

symbole de la transition et n la valeur de l'état. Et étant donné que pour chaque couple (b, m) il ne peut n'avoir que ce couple avec comme seconde valeur m alors la transition vers cet état sera toujours la même.

5. Les automates de Glushkov sont des hamacs. Car ils sont standard, accessible et coaccessible. Et que toutes leurs orbites maximales sont fortement stables et transversales.

3 Conclusion

L'algorithme de Glushkov permet de convertir une expression régulière en automate. Avec les expressions régulières, on peut simplement décrire un langage et avec les automates, on peut simplement savoir si un mot est reconnu. Il est très utilisé en *informatique*, parce que pour les humains, il est plus simple de décrire un langage avec une expression régulière. Et les machines comprennent très facilement les automates. Ce qui fait qu'il est possible de faire des *programmes informatiques* qui reconnaissent un langage et exécutent des tâches à chaque mot.

V Conclusion

Ce document a fourni une définition formelle des concepts clefs utilisés dans la théorie des langages et les automates. Nous avons d'abord introduit les notions de mots, de langages et les opérations fondamentales qui leur sont associées. Ensuite, nous avons exploré les expressions régulières, leurs définitions et les fonctions qui peuvent être appliquées sur elles. Par la suite, nous avons étudié les automates, en particulier ceux sans transitions ε, et les fonctions qui leur sont appliquées. Enfin, nous avons abordé les automates de Glushkov, décrivant leur construction et leurs propriétés.

Bien que nous n'ayons couvert qu'une partie des concepts et des opérations possibles, cette introduction vise à fournir une base solide pour comprendre et utiliser ces outils puissants. Pour approfondir vos connaissances, nous vous encourageons à consulter des ressources supplémentaires en théorie des langages et des automates.

Bibliographie

[Kle51] Stephen Cole Kleene. « Representation of Events in Nerve Nets and Finite Automata ». In : 1951.

[Glu61] V M Glushkov. « THE ABSTRACT THEORY OF AUTOMATA ». In : *Russian Mathematical Surveys* 16.5 (oct. 1961), p. 1. DOI : 10.1070/RM1961v016n05ABEH004112. URL : https://dx.doi.org/10.1070/RM1961v016n05ABEH004112.

[Tho68] Ken Thompson. « Programming techniques : Regular expression search algorithm ». In : *Communications of the ACM* 11.6 (1968), p. 419-422.

[Har78] Michael A. Harrison. *Introduction to formal language theory*. English. Addison-Wesley series in computer science. Reading, Mass. : Addison-Wesley Pub. Co., 1978. ISBN : 0201029553 ; 9780201029550.

[Aut94] Jean-Michel Autebert. *Théorie des langages et des automates*. French. Manuels informatiques Masson. Paris : Masson, 1994. ISBN : 2225840016 ; 9782225840012.

[CZ00] Pascal Caron et Djelloul Ziadi. « Characterization of Glushkov automata ». In : *Theor. Comput. Sci.* 233.1-2 (2000), p. 75-90.

[HMU07] 1939- Hopcroft John E., Rajeev Motwani et 1942- Ullman Jeffrey D. *Introduction to automata theory, languages, and computation*. English. 3rd ed. Boston : Pearson/Addison Wesley, 2007. ISBN : 0321455363 ; 9780321455369 ; 0321462254 ; 9780321462251 ; 0321455371 ; 9780321455376 ; 0321476174 ; 9780321476173.

[Car23] Pascal Caron. « Cours de théorie des langages ». Non publié. Document interne à l'Université de Rouen. Déc. 2023.